GCC 中文手册

GCC

Section: GNU Tools (1)
Updated: 2003/12/05
Index Return to Main Contents

NAME

gcc,g++-GNU 工程的 C 和 C++编译器(egcs-1.1.2)

总览(SYNOPSIS)

gcc[*option*|*filename*]...
g++[*option*|*filename*]...

警告(WARNING)

本手册页内容摘自 GNU C 编译器的完整文档,仅限于解释选项的含义.

除非有人自愿维护,否则本手册页不再更新.如果发现手册页和软件之间有所矛盾,请查对 Info 文件, Info 文件是权威文档.

如果我们发觉本手册页的内容由于过时而导致明显的混乱和抱怨时,我们就停止发布它.不可能有其他选择,象更新 Info 文件同时更新 man 手册,因为其他维护 GNU CC 的工作没有留给我们时间做这个. GNU 工程认为 man 手册是过时产物,应该把时间用到别的地方.

如果需要完整和最新的文档,请查阅 Info 文件`**gcc**'或*Using and Porting GNU CC (for version 2.0) (使用和移植 GNU CC 2.0)* 手册.二者均来自 Texinfo 原文件 **gcc.texinfo**.

描述(DESCRIPTION)

C 和 C++编译器是集成的.他们都要用四个步骤中的一个或多个处理输入文件: 预处理(preprocessing),编译(compilation),汇编(assembly)和连接(linking).源文件后缀名标识源文件的 语言,但是对编译器来说,后缀名控制着缺省设定:

gcc

认为预处理后的文件(**.i**)是 C 文件,并且设定 C 形式的连接.

g++

认为预处理后的文件(**.i**)是 C++文件,并且设定 C++形式的连接.

源文件后缀名指出语言种类以及后期的操作:

.c C 源程序;预处理,编译,汇编

.C C++源程序;预处理,编译,汇编

.cc C++源程序;预处理,编译,汇编

.cxx C++源程序;预处理,编译,汇编

.m Objective-C 源程序;预处理,编译,汇编

.i 预处理后的 C 文件;编译,汇编

.ii 预处理后的 C++文件;编译,汇编

.s 汇编语言源程序;汇编

.S 汇编语言源程序;预处理,汇编

.h 预处理器文件;通常不出现在命令行上

其他后缀名的文件被传递给连接器(linker).通常包括:

.o 目标文件(Object file)

.a 归档库文件(Archive file)

除非使用了**-c**, **-S**,或**-E** 选项(或者编译错误阻止了完整的过程),否则连接总是 最后的步骤.在连接阶段中,所有对应于源程序的**.o** 文件, **-l** 库文件,无法识别的文件名(包括指定的

.o 目标文件和**.a** 库文件)按命令行中的顺序传递给连接器.

选项(OPTIONS)

选项必须分立给出: `**-dr**'完全不同于`**-d -r** '.

大多数`**-f**'和`**-W**'选项有两个相反的格式: **-f***name* 和 **-fno-***name* (或**-W***name* 和**-Wno-***name*).这里 只列举不是默认选项的格式.

下面是所有选项的摘要,按类型分组,解释放在后面的章节中.

总体选项(Overall Option)

-c -S -E -o *file* -pipe -v -x *language*

语言选项(Language Option)

-ansi -fall-virtual -fcond-mismatch -fdollars-in-identifiers -fenum-int-equiv -fexternal-templates -fno-asm -fno-builtin -fhosted -fno-hosted -ffreestanding -fno-freestanding -fno-strict-prototype -fsigned-bitfields -fsigned-char -fthis-is-variable -funsigned-bitfields -funsigned-char -fwritable-strings -traditional -traditional-cpp -trigraphs

警告选项(Warning Option)

-fsyntax-only -pedantic -pedantic-errors -w -W -Wall -Waggregate-return -Wcast-align -Wcast-qual -Wchar-subscript -Wcomment -Wconversion -Wenum-clash -Werror -Wformat -Wid-clash-*len* -Wimplicit -Wimplicit-int -Wimplicit-function-declaration -Winline -Wlong-long -Wmain -Wmissing-prototypes -Wmissing-declarations -Wnested-externs -Wno-import -Wparentheses -Wpointer-arith -Wredundant-decls -Wreturn-type -Wshadow -Wstrict-prototypes -Wswitch -Wtemplate-debugging -Wtraditional -Wtrigraphs -Wuninitialized -Wunused -Wwrite-strings

调试选项(Debugging Option)

-a -d*letters* -fpretend-float -g -g*level* -gcoff -gxcoff -gxcoff+ -gdwarf -gdwarf+ -gstabs -gstabs+ -ggdb -p -pg -save-temps -print-file-name=*library* -print-libgcc-file-name -print-prog-na

me=*program*

优化选项(Optimization Option)

-fcaller-saves -fcse-follow-jumps -fcse-skip-blocks -fdelayed-branch -felide-constructors -fexpensive-optimizations -ffast-math -ffloat-store -fforce-addr -fforce-mem -finline-functions -fkeep-inline-functions -fmemoize-lookups -fno-default-inline -fno-defer-pop -fno-function-cse -fno-inline -fno-peephole -fomit-frame-pointer -frerun-cse-after-loop -fschedule-insns -fschedule-insns2 -fstrength-reduce -fthread-jumps -funroll-all-loops -funroll-loops -O -O2 -O3

预处理器选项(Preprocessor Option)

-A*assertion* -C -dD -dM -dN -D*macro*[=*defn*] -E -H -idirafter *dir* -include *file* -imacros *file* -iprefix *file* -iwithprefix *dir* -M -MD -MM -MMD -nostdinc -P -U*macro* -undef

汇编器选项(Assembler Option)

-Wa,*option*

连接器选项(Linker Option)

-l*library* -nostartfiles -nostdlib -static -shared -symbolic -Xlinker *option* -Wl,*option* -u *symbol*

目录选项(Directory Option)

-B*prefix* -I*dir* -I- -L*dir*

目标机选项(Target Option)

-b *machine* -V *version*

配置相关选项(Configuration Dependent Option)

M680x0 选项

-m68000 -m68020 -m68020-40 -m68030 -m68040 -m68881 -mbitfield -mc68000 -mc68020 -mfpa -mnobitfield -mrtd -mshort -msoft-float

VAX 选项

-mg -mgnu -munix

SPARC 选项

-mepilogue -mfpu -mhard-float -mno-fpu -mno-epilogue -msoft-fl

oat -msparclite -mv8 -msupersparc -mcypress

Convex 选项

-margcount -mc1 -mc2 -mnoargcount

AMD29K 选项

-m29000 -m29050 -mbw -mdw -mkernel-registers -mlarge -mnbw -mnodw -msmall -mstack-check -muser-registers

M88K 选项

-m88000 -m88100 -m88110 -mbig-pic -mcheck-zero-division -mhandle-large-shift -midentify-revision -mno-check-zero-division -mno-ocs-debug-info -mno-ocs-frame-position -mno-optimize-arg-area -mno-serialize-volatile -mno-underscores -mocs-debug-info -mocs-frame-position -moptimize-arg-area -mserialize-volatile -mshort-data-*num* -msvr3 -msvr4 -mtrap-large-shift -muse-div-instruction -mversion-03.00 -mwarn-passed-structs

RS6000 选项

-mfp-in-toc -mno-fop-in-toc

RT 选项

-mcall-lib-mul -mfp-arg-in-fpregs -mfp-arg-in-gregs -mfull-fp-blocks -mhc-struct-return -min-line-mul -mminimum-fp-blocks -mnohc-struct-return

MIPS 选项

-mcpu=*cpu type* -mips2 -mips3 -mint64 -mlong64 -mmips-as -mgas -mrnames -mno-rnames -mgpopt -mno-gpopt -mstats -mno-stats -mmemcpy -mno-memcpy -mno-mips-tfile -mmips-tfile -msoft-float -mhard-float -mabicalls -mno-abicalls -mhalf-pic -mno-half-pic -G *num* -nocpp

i386 选项

-m486 -mno-486 -msoft-float -mno-fp-ret-in-387

HPPA 选项

-mpa-risc-1-0 -mpa-risc-1-1 -mkernel -mshared-libs -mno-shared-libs -mlong-calls -mdisable-fpregs -mdisable-indexing -mtrailing-colon

i960 选项

-m*cpu-type* -mnumerics -msoft-float -mleaf-procedures -mno-leaf-procedures -mtail-call -mno-tail-call -mcomplex-addr -mno-complex-addr -mcode-align -mno-code-align -mic-compat -mic2.0-compat -mic3.0-compat -masm-compat -mintel-asm -mstrict-align -mno-strict-align -mold-align -mno-old-align

DEC Alpha 选项

-mfp-regs -mno-fp-regs -mno-soft-float -msoft-float

System V 选项

-G -Qy -Qn -YP,*paths* -Ym,*dir*

代码生成选项(Code Generation Option)

-fcall-saved-*reg* -fcall-used-*reg* -ffixed-*reg* -finhibit-size-directive -fnonnull-objects -fno-common -fno-ident -fno-gnu-linker -fpcc-struct-return -fpic -fPIC -freg-struct-return -fshared-data -fshort-enums -fshort-double -fvolatile -fvolatile-global -fverbose-asm

总体选项(Overall Option)

-x *language*

明确指出后面输入文件的语言为 *language* (而不是从文件名后缀得到的默认选择). 这个选项应用于后面 所有的输入文件,直到遇着下一个`**-x**'选项. *language* 的可选值有`**c**', `**objective-c**', `**c-header**', `**c++**', `**cpp-output**', `**assembler**',和`**assembler-with-cpp**'.

-x none

关闭任何对语种的明确说明,因此依据文件名后缀处理后面的文件(就象是从未使用过

`**-x**'选项).

如果只操作四个阶段(预处理,编译,汇编,连接)中的一部分,可以使用`**-x**'选项(或文件名后缀)告诉 **gcc** 从哪里开始,用`**-c**', `**-S**',或`**-E**'选项告诉 **gcc** 到哪里结束.注意,某些选项组合(例如, `**-x cpp-output -E**')使 **gcc** 不作任何事情.

-c

编译或汇编源文件,但是不作连接.编译器输出对应于源文件的目标文件.

缺省情况下, GCC 通过用`**.o**'替换源文件名后缀`**.c**', `**.i**', `**.s**',等等,产生目标文件名.可以使用**-o** 选项选择其他名字.

GCC 忽略**-c** 选项后面任何无法识别的输入文件(他们不需要编译或汇编).

-S

编译后即停止,不进行汇编.对于每个输入的非汇编语言文件,输出文件是汇编语言文件.

缺省情况下, GCC 通过用`**.o**'替换源文件名后缀`**.c**', `**.i**',等等,产生 目标文件名.可以使用**-o** 选项选择其他名字.

GCC 忽略任何不需要编译的输入文件.

-E

预处理后即停止,不进行编译.预处理后的代码送往标准输出.

GCC 忽略任何不需要预处理的输入文件.

-o *file*

指定输出文件为 *file*.该选项不在乎 GCC 产生什么输出,无论是可执行文件,目标文件,汇编文件还是 预处理后的 C 代码.

由于只能指定一个输出文件,因此编译多个输入文件时,使用`**-o**'选项没有意义,除非输出一个可执行文件.

如果没有使用`**-o**'选项,默认的输出结果是:可执行文件为`**a.out**', `*source*.*suffix* '的目标文件是`*source*.**o**',汇编文件是 `*source*.**s**',而预处理后的 C 源代码送往标准输出.

-v

(在标准错误)显示执行编译阶段的命令.同时显示编译器驱动程序,预处理器,编译器的版本号.

-pipe

在编译过程的不同阶段间使用管道而非临时文件进行通信.这个选项在某些系统上无法工作,因为那些系统的 汇编器不能从管道读取数据. GNU 的汇编器没有这个问题.

语言选项(LANGUAGE OPTIONS)

下列选项控制编译器能够接受的 C "方言":

-ansi

支持符合 ANSI 标准的 C 程序.

这样就会关闭 GNU C 中某些不兼容 ANSI C 的特性,例如 **asm**, **inline** 和 **typeof** 关键字,以及诸如 **unix** 和 **vax** 这些表明当前系统类型的预定义宏.同时开启 不受欢迎和极少使用的 ANSI trigraph 特性,以及禁止`**$**'成为标识符的一部分.

尽管使用了`**-ansi**'选项,下面这些可选的关键字, **__asm__**, **__extension__**, **__inline__**和**__typeof__**仍然有效.你当然不会把 他们用在 ANSI C 程序中,但可以把他们放在头文件里,因为编译包含这些头文件的程序时,可能会指定 `**-ansi**'选项.另外一些预定义宏,如**__unix__**和**__vax__**,无论有没有使用 `**-ansi**'选项,始终有效.

使用`**-ansi**'选项不会自动拒绝编译非 ANSI 程序,除非增加`**-pedantic**'选项作为`**-ansi**'选项的补充.

使用`**-ansi**'选项的时候,预处理器会预定义一个**__STRICT_ANSI__**宏.有些头文件关注此宏,以避免声明某些函数,或者避免定义某些宏,这些函数和宏不被 ANSI 标准调用;这样就不会干扰在其他地方 使用这些名字的程序了.

-fno-asm

不把 **asm**, **inline** 或 **typeof** 当作关键字,因此这些词可以用做标识符.用 **__asm__**, **__inline__**和**__typeof__**能够替代他们. `**-ansi**' 隐含声明了`**-fno-asm**'.

-fno-builtin

不接受不是两个下划线开头的内建函数(built-in function).目前受影响的函数有**_exit**, **abort**, **abs**, **alloca**, **cos**, **exit**, **fabs**, **labs**, **memcmp**, **memcpy**, **sin**, **sqrt**, **strcmp**, **strcpy**,和 **strlen**.

`**-ansi**'选项能够阻止 **alloca** 和**_exit** 成为内建函数.

-fhosted

按宿主环境编译;他隐含声明了`**-fbuiltin**'选项,而且警告不正确的 **main** 函数声明.

-ffreestanding

按独立环境编译;他隐含声明了`**-fno-builtin**'选项,而且对 **main** 函数没有特别要求.

(译注:宿主环境(hosted environment)下所有的标准库可用, main 函数返回一个 int 值,典型例子是除了 内核以外几乎所有的程序.对应的独立环境(freestanding environment)不存在标准库,程序入口也不一定是 main,最明显的例子就是操作系统内核.详情参考 gcc 网站最近的资料)

-fno-strict-prototype

对于没有参数的函数声明,例如`**int foo** ();',按 C 风格处理---即不说明参数个数或类型. (仅针对 C++).正常情况下,这样的函数 **foo** 在 C++中意味着参数为空.

-trigraphs

支持 ANSI C trigraphs. `**-ansi**'选项隐含声明了`**-trigraphs**'.

-traditional

试图支持传统 C 编译器的某些方面.详见 GNU C 手册,我们已经把细节清单从这里删除,这样当内容过时后,人们也不会 埋怨我们.

除了一件事:对于 C++程序(不是 C), `**-traditional**'选项带来一个附加效应,允

许对 **this** 赋值.他和`**-fthis-is-variable**'选项的效果一样.

-traditional-cpp

试图支持传统 C 预处理器的某些方面.特别是上面提到有关预处理器的内容,但是不包括 `**-traditional**'选项的其他效应.

-fdollars-in-identifiers

允许在标识符(identifier)中使用`**$**'字符(仅针对 C++).你可以指定 `**-fno-dollars-in-identifiers**'选项显明禁止使用`**$**'符. (GNU C++在某些 目标系统缺省允许`**$**'符,但不是所有系统.)

-fenum-int-equiv

允许 **int** 类型到枚举类型(enumeration)的隐式转换(仅限于 C++).正常情况下 GNU C++允许从 **enum** 到 **int** 的转换,反之则不行.

-fexternal-templates

为模板声明(template declaration)产生较小的代码(仅限于 C++),方法是对于每个模板函数 (template function),只在定义他们的地方生成一个副本.想要成功使用这个选项,你必须在所有使用模板的 文件中,标记`**#pragma implementation**' (定义)或`**#pragma interface**' (声明).

当程序用`**-fexternal-templates**'编译时,模板实例(template instantiation) 全部是外部类型.你必须让需要的实例在实现文件中出现.可以通过 **typedef** 实现这一点,他引用所需的每个 实例.相对应的,如果编译时使用缺省选项`**-fno-external-templates**',所有模板实例明确的设为内置.

-fall-virtual

所有可能的成员函数默认为虚函数.所有的成员函数(除了构造子函数和 **new** 或 **delete** 成员操作符)视为所在类的虚函数.

这不表明每次调用成员函数都将通过内部虚函数表.有些情况下,编译器能够判断出可以直接调用某个虚函数;这时就 直接调用.

-fcond-mismatch

允许条件表达式的第二和第三个参数的类型不匹配.这种表达式的值是 void.

-fthis-is-variable

允许对 **this** 赋值(仅对 C++).合并用户自定义的自由存储管理机制到 C++后,使可赋

值的 `**this**'显得不合时宜.因此,默认情况下,类成员函数内部对 **this** 赋值是无效操作.然而为了 向后兼容,你可以通过`**-fthis-is-variable**'选项使这种操作有效.

-funsigned-char

把 **char** 定义为无符号类型,如同 **unsigned char**.

各种机器都有自己缺省的 **char** 类型.既可能是 **unsigned char** 也可能是 **signed char** .

理想情况下,当依赖于数据的符号性时,一个可移植程序总是应该使用 **signed char** 或 **unsigned char**.但是许多程序已经写成只用简单的 **char**,并且期待这是有符号数(或者无符号数,具体情况取决于 编写程序的目标机器).这个选项,和它的反义选项,使那样的程序工作在对应的默认值上.

char 的类型始终应该明确定义为 **signed char** 或 **unsigned char**,即使 它表现的和其中之一完全一样.

-fsigned-char

把 **char** 定义为有符号类型,如同 **signed char**.

这个选项等同于`**-fno-unsigned-char**',他是 the negative form of `**-funsigned-char**'的相反选项.同样, `**-fno-signed-char**'等价于`**-funsigned-char**'.

-fsigned-bitfields
-funsigned-bitfields
-fno-signed-bitfields
-fno-unsigned-bitfields

如果没有明确声明`**signed**'或`**unsigned**'修饰符,这些选项用来定义有符号位域(bitfield)或无符号位域.缺省情况下,位域是有符号的,因为他们继承的基本整数类型,如 **int**,是 有符号数.

然而,如果指定了`**-traditional**'选项,位域永远是无符号数.

-fwritable-strings

把字符串常量存储到可写数据段,而且不做特别对待.这是为了兼容一些老程序,他们

假设字符串常量是可写的. `**-traditional**'选项也有相同效果.

篡改字符串常量是一个非常糟糕的想法; ``常量''就应该是常量.

预处理器选项(Preprocessor Option)

下列选项针对 C 预处理器,预处理器用在正式编译以前,对 C 源文件进行某种处理.

如果指定了`**-E**'选项, GCC 只进行预处理工作.下面的某些选项必须和`**-E**'选项一起才 有意义,因为他们的输出结果不能用于编译.

-include *file*

在处理常规输入文件之前,首先处理文件 *file*,其结果是,文件 *file* 的内容先得到编译. 命令行上任何`**-D**'和`**-U**'选项永远在`**-include** *file*'之前处理, 无论他们在命令行上的顺序如何.然而`**-include**'和`**-imacros**'选项按书写顺序处理.

-imacros *file*

在处理常规输入文件之前,首先处理文件 *file*,但是忽略输出结果.由于丢弃了文件 *file* 的 输出内容, `**-imacros** *file*'选项的唯一效果就是使文件 *file* 中的宏定义生效, 可以用于其他输入文件.在处理`**-imacros***file*'选项之前,预处理器首先处理`**-D**' 和`**-U**'选项,并不在乎他们在命令行上的顺序.然而`**-include**'和 `**-imacros**'选项按书写顺序处理.

-idirafter *dir*

把目录 *dir* 添加到第二包含路径中.如果某个头文件在主包含路径(用`**-I**'添加的路径)中没有 找到,预处理器就搜索第二包含路径.

-iprefix *prefix*

指定 *prefix* 作为后续`**-iwithprefix**'选项的前缀.

-iwithprefix *dir*

把目录添加到第二包含路径中.目录名由 *prefix* 和 *dir* 合并而成,这里 *prefix* 被先前的`**-iprefix**'选项指定.

-nostdinc

不要在标准系统目录中寻找头文件.只搜索`**-I**'选项指定的目录(以及当前目录,如果合适).

结合使用`**-nostdinc**'和`**-I-**'选项,你可以把包含文件搜索限制在显式指定的目录.

-nostdinc++

不要在 C++专用标准目录中寻找头文件,但是仍然搜索其他标准目录. (当建立`**libg++**'时使用 这个选项.)

-undef

不要预定义任何非标准宏. (包括系统结构标志).

-E

仅运行 C 预处理器.预处理所有指定的 C 源文件,结果送往标准输出或指定的输出文件.

-C

告诉预处理器不要丢弃注释.配合`**-E**'选项使用.

-P

告诉预处理器不要产生`**#line**'命令.配合`**-E**'选项使用.

-M [-MG]

告诉预处理器输出一个适合 **make** 的规则,用于描述各目标文件的依赖关系.对于每个源文件,预处理器输出 一个 **make** 规则,该规则的目标项(target)是源文件对应的目标文件名,依赖项(dependency)是源文件中 `**#include** 引用的所有文件.生成的规则可以是单行,但如果太长,就用`****'-换行符续成多行.规则 显示在标准输出,不产生预处理过的 C 程序.

`**-M**'隐含了`**-E**'选项.

`**-MG**'要求把缺失的头文件按存在对待,并且假定他们和源程序文件在同一目录下.必须和 `**-M**'选项一起用.

-MM [-MG]

和`**-M**'选项类似,但是输出结果仅涉及用户头文件,象这样`**#include** *file*"'.忽略系统头文件如`**#include** <*file*>'.

-MD

和`**-M**'选项类似,但是把依赖信息输出在文件中,文件名通过把输出文件名末尾的`**.o**'替换为 `**.d**'产生.同时继续指定的编译工作---`**-MD**'不象`**-M**'那样阻止正常的编

译任务.

Mach 的实用工具`**md**'能够合并`**.d**'文件,产生适用于`**make**'命令的单一的 依赖文件.

-MMD

和`**-MD**'选项类似,但是输出结果仅涉及用户头文件,忽略系统头文件.

-H

除了其他普通的操作, GCC 显示引用过的头文件名.

-A*question***(***answer***)**

如果预处理器做条件测试,如`**#if #***question***(***answer***)**',该选项可以断言(Assert) *question* 的答案是 *answer*. **-A-**'关闭一般用于描述目标机的标准断言.

-D*macro*

定义宏 *macro*,宏的内容定义为字符串`**1**'.

-D*macro***=***defn*

定义宏 *macro* 的内容为 *defn*.命令行上所有的`**-D**'选项在 `**-U**'选项之前处理.

-U*macro*

取消宏 *macro*. `**-U**'选项在所有的`**-D**'选项之后处理,但是优先于任何 `**-include**'或`**-imacros**'选项.

-dM

告诉预处理器输出有效的宏定义列表(预处理结束时仍然有效的宏定义).该选项需结合`**-E**'选项使用.

-dD

告诉预处理器把所有的宏定义传递到输出端,按照出现的顺序显示.

-dN

和`**-dD**'选项类似,但是忽略宏的参量或内容.只在输出端显示`**#define** *name*.

汇编器选项(ASSEMBLER OPTION)

-Wa,*option*

把选项 *option* 传递给汇编器.如果 *option* 含有逗号,就在逗号处分割成多个选项.

连接器选项(LINKER OPTION)

下面的选项用于编译器连接目标文件,输出可执行文件的时候.如果编译器不进行 连接,他们就毫无意义.

object-file-name

如果某些文件没有特别明确的后缀a special recognized suffix, GCC就认为他们是目标文件或库文件. (根据文件内容,连接器能够区分目标文件和库文件).如果GCC执行连接操作,这些目标文件将成为连接器的输入文件.

-l*library*

连接名为 *library* 的库文件.

连接器在标准搜索目录中寻找这个库文件,库文件的真正名字是`**lib***library*.a'.连接器会 当做文件名得到准确说明一样引用这个文件.

搜索目录除了一些系统标准目录外,还包括用户以`**-L**'选项指定的路径.

一般说来用这个方法找到的文件是库文件---即由目标文件组成的归档文件(archive file).连接器处理归档文件的 方法是:扫描归档文件,寻找某些成员,这些成员的符号目前已被引用,不过还没有被定义.但是,如果连接器找到普通的 目标文件,而不是库文件,就把这个目标文件按平常方式连接进来.指定`**-l**'选项和指定文件名的唯一区别是, `**-l** 选项用`**lib**'和`**.a**'把 *library* 包裹起来,而且搜索一些目录.

-lobjc

这个**-l**选项的特殊形式用于连接Objective C程序.

-nostartfiles

不连接系统标准启动文件,而标准库文件仍然正常使用.

-nostdlib

不连接系统标准启动文件和标准库文件.只把指定的文件传递给连接器.

-static

在支持动态连接(dynamic linking)的系统上,阻止连接共享库.该选项在其他系统上无效.

-shared

生成一个共享目标文件,他可以和其他目标文件连接产生可执行文件.只有部分系统支

持该选项.

-symbolic

建立共享目标文件的时候,把引用绑定到全局符号上.对所有无法解析的引用作出警告(除非用连接编辑选项 `**-Xlinker -z -Xlinker defs**'取代).只有部分系统支持该选项.

-Xlinker *option*

把选项 *option* 传递给连接器.可以用他传递系统特定的连接选项, GNU CC 无法识别这些选项.

如果需要传递携带参数的选项,你必须使用两次`**-Xlinker**',一次传递选项,另一次传递他的参数. 例如,如果传递`**-assert definitions**',你必须写成`**-Xlinker -assert -Xlinker definitions**',而不能写成`**-Xlinker "-assert definitions"**',因为这样会把整个 字符串当做一个参数传递,显然这不是连接器期待的.

-Wl,*option*

把选项 *option* 传递给连接器.如果 *option* 中含有逗号,就在逗号处分割成多个选项.

-u *symbol*

使连接器认为取消了 *symbol* 的符号定义,从而连接库模块以取得定义.你可以使用多个 `**-u**'选项,各自跟上不同的符号,使得连接器调入附加的库模块.

目录选项(DIRECTORY OPTION)

下列选项指定搜索路径,用于查找头文件,库文件,或编译器的某些成员:

-I*dir*

在头文件的搜索路径列表中添加 *dir* 目录.

-I-

任何在`**-I-**'前面用`**-I**'选项指定的搜索路径只适用于`**#include "***file***"**'这种情况;他们不能用来搜索`**#include <***file***>**'包含的头文件.

如果用`**-I**'选项指定的搜索路径位于`**-I-**'选项后面,就可以在这些路径中搜索所有的 `**#include**'指令. (一般说来**-I** 选项就是这么用的.)

还有, `**-I-**'选项能够阻止当前目录(存放当前输入文件的地方)成为搜索`**#include "*file*"**'的第一选择.没有办法克服`**-I-**'选项的这个效应.你可以指定 `**-I.**'搜索那个目录,它在调用编译器时是当前目录.这和预处理器的默认行为不完全一样,但是结果通常 令人满意.

`**-I-**'不影响使用系统标准目录,因此, `**-I-**'和`**-nostdinc**'是不同的选项.

-L*dir*

在`**-l**'选项的搜索路径列表中添加 *dir* 目录.

-B*prefix*

这个选项指出在何处寻找可执行文件,库文件,以及编译器自己的数据文件.

编译器驱动程序需要执行某些下面的子程序: `**cpp**', `**cc1**' (或 C++的 `**cc1plus**'), `**as**'和`**ld**'.他把 *prefix* 当作欲执行的程序的 前缀,既可以包括也可以不包括`*machine/version/*'.

对于要运行的子程序,编译器驱动程序首先试着加上`**-B**'前缀(如果存在).如果没有找到文件,或没有指定 `**-B**'选项,编译器接着会试验两个标准前缀`**/usr/lib/gcc/**'和 `**/usr/local/lib/gcc-lib/**'.如果仍然没能够找到所需文件,编译器就在`**PATH**'环境变量 指定的路径中寻找没加任何前缀的文件名.

如果有需要,运行时(run-time)支持文件`**libgcc.a**'也在`**-B**'前缀的搜索范围之内. 如果这里没有找到,就在上面提到的两个标准前缀中寻找,仅此而已.如果上述方法没有找到这个文件,就不连接他了.多数 情况的多数机器上, `**libgcc.a**'并非必不可少.

你可以通过环境变量 **GCC_EXEC_PREFIX** 获得近似的效果;如果定义了这个变量,其值就和上面说的 一样用做前缀.如果同时指定了`**-B**'选项和 **GCC_EXEC_PREFIX** 变量,编译器首先使用 `**-B**'选项,然后才尝试环境变量值.

警告选项(WARNING OPTION)

警告是针对程序结构的诊断信息,程序不一定有错误,而是存在风险,或者可能存在 错误.

下列选项控制 GNU CC 产生的警告的数量和类型:

-fsyntax-only

检查程序中的语法错误,但是不产生输出信息.

-w

禁止所有警告信息.

-Wno-import

禁止所有关于**#import** 的警告信息.

-pedantic

打开完全服从 ANSI C 标准所需的全部警告诊断;拒绝接受采用了被禁止的语法扩展的程序.

无论有没有这个选项,符合 ANSI C 标准的程序应该能够被正确编译(虽然极少数程序需要`**-ansi**' 选项).然而,如果没有这个选项,某些 GNU 扩展和传统 C 特性也得到支持.使用这个选项可以拒绝这些程序.没有理由 使用这个选项,他存在只是为了满足一些书呆子(pedant).

对于替选关键字(他们以`__'开始和结束) `**-pedantic**'不会产生警告信息. Pedantic 也不警告跟在**__extension__**后面的表达式.不过只应该在系统头文件中使用这种转义措施,应用程序最好 避免.

-pedantic-errors

该选项和`**-pedantic**'类似,但是显示错误而不是警告.

-W

对下列事件显示额外的警告信息:

*

非易变自动变量(nonvolatile automatic variable)可能在调用 **longjmp** 时发生改变. 这些警告仅在优化编译时发生.

编译器只知道对 **setjmp** 的调用,他不可能知道会在哪里调用 **longjmp**,事实上一个信号处理例程可以在程序的任何地点调用他.其结果是,即使程序没有问题,你也可能

会得到警告,因为无法在可能出现问题 的地方调用 **longjmp.**

*

既可以返回值,也可以不返回值的函数. (缺少结尾的函数体被看作不返回函数值)例如,下面的函数将导致这种警告:

```
foo (a)
{
  if (a > 0)
    return a;
}
```

由于 GNU CC 不知道某些函数永不返回(含有 **abort** 和 **longjmp**),因此有可能出现虚假警告.

*

表达式语句或逗号表达式的左侧没有产生作用(side effect).如果要防止这种警告,应该把未使用的表达式强制转换 为 void 类型.例如,这样的表达式`**x[i,j]**'会导致警告,而`**x[(void)i,j]**'就不会.

*

无符号数用`**>**'或`**<=**'和零做比较.

-Wimplicit-int

警告没有指定类型的声明.

-Wimplicit-function-declaration

警告在声明之前就使用的函数.

-Wimplicit

同-Wimplicit-int 和-Wimplicit-function-declaration.

-Wmain

如果把 **main** 函数声明或定义成奇怪的类型,编译器就发出警告.典型情况下,这个函数用于外部连接, 返回 **int** 数值,不需要参数,或指定两个参数.

-Wreturn-type

如果函数定义了返回类型,而默认类型是 **int** 型,编译器就发出警告.同时警告那些不带返回值的 **return** 语句,如果他们所属的函数并非 **void** 类型.

-Wunused

如果某个局部变量除了声明就没再使用,或者声明了静态函数但是没有定义,或者某条语句的运算结果显然没有使用, 编译器就发出警告.

-Wswitch

如果某条 **switch** 语句的参数属于枚举类型,但是没有对应的 **case** 语句使用枚举元素,编译器 就发出警告. (**default** 语句的出现能够防止这个警告.)超出枚举范围的 **case** 语句同样会 导致这个警告.

-Wcomment

如果注释起始序列`**/***'出现在注释中,编译器就发出警告.

-Wtrigraphs

警告任何出现的 trigraph (假设允许使用他们).

-Wformat

检查对 **printf** 和 **scanf** 等函数的调用,确认各个参数类型和格式串中的一致.

-Wchar-subscripts

警告类型是 **char** 的数组下标.这是常见错误,程序员经常忘记在某些机器上 char 有符号.

-Wuninitialized

在初始化之前就使用自动变量.

这些警告只可能做优化编译时出现,因为他们需要数据流信息,只有做优化的时候才估算数据流信息.如果不指定 `**-O**'选项,就不会出现这些警告.

这些警告仅针对等候分配寄存器的变量.因此不会发生在声明为 **volatile** 的变量上面,不会发生在已经 取得地址的变量,或长度不等于 1, 2, 4, 8 字节的变量.同样也不会发生在结构,联合或数组上面,即使他们在 寄存器中.

注意,如果某个变量只计算了一个从未使用过的值,这里可能不会警告.因为在显示警告之前,这样的计算已经被 数据流分析删除了.

这些警告作为可选项是因为 GNU CC 还没有智能到判别所有的情况,知道有些看上去错误的代码其实是正确的.下面是 一个这样的例子:

```
{
  int x;
  switch (y)
```

```
    {
    case 1: x = 1;
      break;
    case 2: x = 4;
      break;
    case 3: x = 5;
    }
  foo (x);
}
```

如果 **y** 始终是 1, 2 或 3,那么 **x** 总会被初始化,但是 GNU CC 不知道这一点.下面是另一个普遍案例:

```
{
  int save_y;
  if (change_y) save_y = y, y = new_y;
  ...
  if (change_y) y = save_y;
}
```

这里没有错误,因为只有设置了 **save_y** 才使用他.

把所有不返回的函数定义为 **volatile** 可以避免某些似是而非的警告.

-Wparentheses

在某些情况下如果忽略了括号,编译器就发出警告.

-Wtemplate-debugging

当在 C++程序中使用 template 的时候,如果调试(debugging)没有完全生效,编译器就发出警告. (仅用于 C++).

-Wall

结合所有上述的`**-W**'选项.通常我们建议避免这些被警告的用法，我们相信,恰当结合宏的使用能够 轻易避免这些用法。

剩下的`**-W...**'选项不包括在`**-Wall**'中,因为我们认为在必要情况下,这些被编译器警告 的

程序结构,可以合理的用在"干净的"程序中.

-Wtraditional

如果某些程序结构在传统 C 中的表现和 ANSI C 不同,编译器就发出警告.

*

宏参出现在宏体的字符串常量内部.传统 C 会替换宏参,而 ANSI C 则视其为常量的一部分.

*

某个函数在块(block)中声明为外部,但在块结束后才调用.

*

switch 语句的操作数类型是 **long**.

-Wshadow

一旦某个局部变量屏蔽了另一个局部变量,编译器就发出警告.

-Wid-clash-*len*

一旦两个确定的标识符具有相同的前 *len* 个字符,编译器就发出警告.他可以协助你开发一些将要在某些 过时的,危害大脑的编译器上编译的程序.

-Wpointer-arith

任何语句如果依赖于函数类型的大小(size)或者 **void** 类型的大小,编译器就发出警告. GNU C 为了 便于计算 **void *** 指针和函数指针,就把这些类型的大小定义为 1.

-Wcast-qual

一旦某个指针强制类型转换以便移除类型修饰符时,编译器就发出警告.例如,如果把 **const char *** 强制转换为普通的 **char *** 时,警告就会出现.

-Wcast-align

一旦某个指针类型强制转换时,导致目标所需的地址对齐(alignment)增加,编译器就发出警告.例如,某些机器上 只能在 2 或 4 字节边界上访问整数,如果在这种机型上把 **char *** 强制转换成 **int *** 类型, 编译器就发出警告.

-Wwrite-strings

规定字符串常量的类型是 **const char[***length***]**,因此,把这样的地址复制给 non-**const char *** 指针将产生警告.这些警告能够帮助你在编译期间发现企图写入字符串常量 的代码,但是你必须非常仔细的在声明和原形中使用 **const**,否则他们只能带来麻烦;所以我们没有让 `**-Wall**' 提供这些警告.

-Wconversion

如果某函数原形导致的类型转换和无函数原形时的类型转换不同,编译器就发出警告.

这里包括定点数和浮点数的 互相转换,改变定点数的宽度或符号,除非他们和缺省声明(default promotion)相同.

-Waggregate-return

如果定义或调用了返回结构或联合的函数,编译器就发出警告. (从语言角度你可以返回一个数组,然而同样会 导致警告.)

-Wstrict-prototypes

如果函数的声明或定义没有指出参数类型,编译器就发出警告. (如果函数的前向引用说明指出了参数类型,则允许后面 使用旧式风格的函数定义,而不会产生警告.)

-Wmissing-prototypes

如果没有预先声明函数原形就定义了全局函数,编译器就发出警告.即使函数定义自身提供了函数原形也会产生这个警告. 他的目的是检查没有在头文件中声明的全局函数.

-Wmissing-declarations

如果没有预先声明就定义了全局函数,编译器就发出警告.即使函数定义自身提供了函数原形也会产生这个警告.这个选项 的目的是检查没有在头文件中声明的全局函数.

-Wredundant-decls

如果在同一个可见域某定义多次声明,编译器就发出警告,即使这些重复声明有效并且毫无差别.

-Wnested-externs

如果某 **extern** 声明出现在函数内部,编译器就发出警告.

-Wenum-clash

对于不同枚举类型之间的转换发出警告(仅适用于 C++).

-Wlong-long

如果使用了 **long long** 类型就发出警告.该警告是缺省项.使用`**-Wno-long-long**' 选项能够防止这个警告. `**-Wlong-long**'和`**-Wno-long-long**'仅在 `**-pedantic**'之下才起作用.

-Woverloaded-virtual

(仅适用于 C++.)在继承类中,虚函数的定义必须匹配虚函数在基类中声明的类型特征(type signature).当 继承类声明了某个函数,它可能是个错误的尝试企图定义一个虚函数,使用这个选项能够产生警告:就是说,当某个函数和基类 中的虚函数同名,但是类型特征不符合基类的任何虚函数,编译器将发出警告.

-Winline

如果某函数不能内嵌(inline),无论是声明为 inline 或者是指定了**-finline-functions** 选项,编译器都将发出警告.

-Werror

视警告为错误;出现任何警告即放弃编译.

调试选项(DEBUGGING OPTION)

GNU CC 拥有许多特别选项,既可以调试用户的程序,也可以对 GCC 排错:

-g

以操作系统的本地格式(stabs, COFF, XCOFF,或 DWARF).产生调试信息. GDB 能够使用这些调试信息.

在大多数使用 stabs 格式的系统上, `**-g**'选项启动只有 GDB 才使用的额外调试信息;这些信息使 GDB 调试效果更好,但是有可能导致其他调试器崩溃,或拒绝读入程序.如果你确定要控制是否生成额外的信息, 使用`**-gstabs+**', `**-gstabs**', `**-gxcoff+**', `**-gxcoff**', `**-gdwarf+**',或`**-gdwarf**' (见下文).

和大多数 C 编译器不同, GNU CC 允许结合使用`**-g**'和`**-O**'选项.优化的代码偶尔制造 一些惊异的结果:某些声明过的变量根本不存在;控制流程直接跑到没有预料到的地方;某些语句因为计算结果是常量或已经确定而 没有执行;某些语句在其他地方执行,因为他们被移到循环外面了.

然而它证明了调试优化的输出是可能的.对可能含有错误的程序使用优化器是合理的.

如果 GNU CC 支持输出多种调试信息,下面的选项则非常有用.

-ggdb

以本地格式(如果支持)输出调试信息,尽可能包括 GDB 扩展.

-gstabs

以 stabs 格式(如果支持)输出调试信息,不包括 GDB 扩展.这是大多数 BSD 系统上 DBX 使用的格式.

-gstabs+

以 stabs 格式(如果支持)输出调试信息,使用只有 GNU 调试器(GDB)理解的 GNU 扩

展.使用这些扩展有可能导致 其他调试器崩溃或拒绝读入程序.

-gcoff

以 COFF 格式(如果支持)输出调试信息.这是在 System V 第四版以前的大多数 System V 系统上 SDB 使用的 格式.

-gxcoff

以 XCOFF 格式(如果支持)输出调试信息.这是 IBM RS/6000 系统上 DBX 调试器使用的格式.

-gxcoff+

以 XCOFF 格式(如果支持)输出调试信息,使用只有 GNU 调试器(GDB)理解的 GNU 扩展.使用这些扩展有可能导致 其他调试器崩溃或拒绝读入程序.

-gdwarf

以 DWARF 格式(如果支持)输出调试信息.这是大多数 System V 第四版系统上 SDB 使用的格式.

-gdwarf+

以 DWARF 格式(如果支持)输出调试信息,使用只有 GNU 调试器(GDB)理解的 GNU 扩展.使用这些扩展有可能导致 其他调试器崩溃或拒绝读入程序.

-g*level*

-ggdb*level*

-gstabs*level*

-gcoff*level* **-gxcoff***level*

-gdwarf*level*

请求生成调试信息,同时用 *level* 指出需要多少信息.默认的 level 值是 2.

Level 1 输出最少量的信息,仅够在不打算调试的程序段内 backtrace.包括函数和外部变量的描述,但是 没有局部变量和行号信息.

Level 3 包含更多的信息,如程序中出现的所有宏定义.当使用`**-g3**'选项的时候,某些调试器支持 宏扩展.

-p

产生额外代码,用于输出 profile 信息,供分析程序 **prof** 使用.

-pg

产生额外代码,用于输出 profile 信息,供分析程序 **gprof** 使用.

-a

产生额外代码,用于输出基本块(basic block)的 profile 信息,它记录各个基本块的执行次数,供诸如 **tcov** 此类的程序分析.但是注意,这个数据格式并非 **tcov** 期待的.最终 GNU **gprof** 将处理这些数据.

-ax

产生额外代码,用于从'bb.in'文件读取基本块的 profile 参数,把 profile 的结果写到'bb.out' 文件. `bb.in'包含一张函数列表.一旦进入列表中的某个函数,profile 操作就开始,离开最外层的函数后, profile 操作就结束.以`-'为前缀名的函数排除在 profile 操作之外.如果函数名不是唯一的,它可以写成 `/path/filename.d:functionname'来澄清. `bb.out'将列出一些有效的文件名.这四个函数名具有 特殊含义: `__bb_jumps__'导致跳转(jump)频率写进`bb.out'. `__bb_trace__'导致基本块序列通过 管道传到`gzip',输出`bbtrace.gz'文件. `__bb_hidecall__'导致从跟踪(trace)中排除 call 指令. `__bb_showret__'导致在跟踪中包括返回指令.

-d*letters*

编译的时候,在 *letters* 指定的时刻做调试转储(dump).用于调试编译器.大多数转储的文件名 通过源文件名添加字词获得(例如`**foo.c.rtl**'或`**foo.c.jump**').

-dM

预处理结束的时候转储所有的宏定义,不输出到文件.

-dN

预处理结束的时候转储所有的宏名.

-dD

预处理结束的时候转储所有的宏定义,同时进行正常输出.

-dy

语法分析(parse)的时候在标准错误转储调试信息.

-dr

RTL 阶段后转储到`*file*.**rtl**'.

-dx

仅对函数生成 RTL,而不是编译.通常和`**r**'联用.

-dj

第一次跳转优化后转储到`*file*.**jump**'.

-ds

CSE (包括有时候跟在 CSE 后面的跳转优化)后转储到`*file*.**cse**'.

-dL

循环优化后转储到`*file*.**loop**'.

-dt

第二次 CSE 处理(包括有时候跟在 CSE 后面的跳转优化)后转储到`*file*.**cse2**'.

-df

流程分析(flow analysis)后转储到`*file*.**flow**'.

-dc

指令组合(instruction combination)后转储到`*file*.**combine**'.

-dS

第一次指令安排(instruction schedule)后转储到`*file*.**sched**'.

-dl

局部寄存器分配后转储到`*file*.**lreg**'.

-dg

全局寄存器分配后转储到`*file*.**greg**'.

-dR

第二次指令安排(instruction schedule)后转储到`*file*.**sched2**'.

-dJ

最后一次跳转优化后转储到`*file*.**jump2**'.

-dd

推迟分支调度(delayed branch scheduling)后转储到`*file*.**dbr**'.

-dk

寄存器-堆栈转换后转储到`*file*.**stack**'.

-da

产生以上所有的转储.

-dm

运行结束后,在标准错误显示内存使用统计.

-dp

在汇编输出加注指明使用了哪些模式(pattern)及其替代模式.

-fpretend-float

交叉编译的时候,假定目标机和宿主机使用同样的浮点格式.它导致输出错误的浮点常数,但是在目标机上运行的时候, 真实的指令序列有可能和 GNU CC 希望的一样.

-save-temps

保存那些通常是``临时''的中间文件;置于当前目录下,并且根据源文件命名.因此,用`**-c -save-temps**'选项编译`**foo.c** '会生成` **foo.cpp**'和`**foo.s**' 以及`**foo.o**'文件.

-print-file-name=*library*

显示库文件 *library* 的全路径名,连接时会使用这个库---其他什么事情都不作.根据这个选项, GNU CC 既不编译,也不连接,仅仅显示文件名.

-print-libgcc-file-name

和`**-print-file-name=libgcc.a**'一样.

-print-prog-name=*program*

类似于`**-print-file-name**',但是查找程序 program 如`**cpp**'.

优化选项(OPTIMIZATION OPTION)

这些选项控制多种优化措施:

-O

-O1

优化.对于大函数,优化编译占用稍微多的时间和相当大的内存.

不使用`**-O**'选项时,编译器的目标是减少编译的开销,使编译结果能够调试.语句是独立的:如果在 两条语句之间用断点中止程序,你可以对任何变量重新赋值,或者在函数体内把程序计数器指到其他语句,以及从源程序中 精确地获取你期待的结果.

不使用`**-O**'选项时,只有声明了 **register** 的变量才分配使用寄存器.编译结果比不用 `**-O**'选项的 PCC 要略逊一筹.

使用了`**-O**'选项,编译器会试图减少目标码的大小和执行时间.

如果指定了`**-O**'选项, `**-fthread-jumps**'和`**-fdefer-pop**'选项将被 打开.在有 delay slot 的机器上, `**-fdelayed-branch**'选项将被打开.在即使没有帧指针 (frame pointer)也支持调试的机器上, `**-fomit-frame-pointer**'选项将被打开.某些机器上 还可能会打开其他选项.

-O2

多优化一些.除了涉及空间和速度交换的优化选项,执行几乎所有的优化工作.例如不进行循环展开(loop unrolling)和函数内嵌(inlining).和**-O** 选项比较,这个选项既增加了编译时间,也提高了生成代码的 运行效果.

-O3

优化的更多.除了打开**-O2** 所做的一切,它还打开了**-finline-functions** 选项.

-O0

不优化.

如果指定了多个**-O** 选项,不管带不带数字,最后一个选项才是生效的选项.

诸如`**-f*flag***'此类的选项描述一些机器无关的开关.大多数开关具有肯定和否定两种格式;`**-ffoo**'开关选项的否定格式应该是`**-fno-foo**'.下面的列表只展示了一种格式---那个不是 默认选项的格式.你可以通过去掉或添加`**no-**'构造出另一种格式.

-ffloat-store

不要在寄存器中存放浮点变量.这样可以防止某些机器上不希望的过高精度,如 68000 的浮点寄存器(来自 68881)保存的精度超过了 **double** 应该具有的精度.

对于大多数程序,过高精度只有好处.但是有些程序严格依赖于 IEEE 浮点数的定义.对这样的程序可以使用 `**-ffloat-store**'选项.

-fmemoize-lookups

-fsave-memoized

使用探索法(heuristic)进行更快的编译(仅对 C++).默认情况下不使用探索法.由于探索法只对某些输入文件 有效,其他程序的编译速度会变得更慢.

第一次编译器必须对成员函数(或对成员数据的引用)建立一个调用.它必须(1)判断出这个类是否实现了那个名字的 成员函数; (2)决定调用哪个成员函数(涉及到推测需要做哪种类型转换); (3)检查成员函数对调用者是否可见.所有这些构成 更慢的编译.一般情形,第二次对成员函数(或对成员数据的引用)建立的调用,必须再次经过相同长度的处理.这意味着象 这样的代码

```
cout << "This " << p << " has " << n << " legs.\n";
```

对整个三步骤要做六次遍历.通过使用软件缓存, ``命中''能够显著地减少这种代价.然而不幸的是,使用这种缓存 必须实现其他机制,带来了它自己的开销. `**-fmemoize-lookups**'选项打开软件缓存.

因为函数的正文环境不同,函数对成员和成员函数的访问权(可见性)也可能不同, **g++**可能需要刷新缓存. 使用`**-fmemoize-lookups**'选项,每编译完一个函数就刷新缓存.而`-fsave-memoized'选项 也启用同样的缓存,但是当编译器发觉最后编译的函数的正文环境产生的访问权和下一个待编译的函数相同,编译器就 保留缓存内容.这对某个类定义许多成员函数时非常有用:除了某些其他类的友函数,每个成员函数拥有和其他成员函数完全一样 的访问权,因而无需刷新缓存.

-fno-default-inline

默认为不要把成员函数内嵌,因为它们定义在类的作用域内(仅 C++).

-fno-defer-pop

一旦函数返回,参数就立即弹出.对于那些调用函数后必须弹出参数的机器,编译器一般情况下让几次函数调用的参数 堆积在栈上,然后一次全部弹出.

-fforce-mem

做数学运算前把将要使用的内存操作数送入寄存器.通过把内存访问转换成潜在的公共子表达式,它可能产生较好的目标码. 如果它们不是公共子表达式,指令组合应该消除各自的寄存器载荷.我乐意倾听不同意见.

-fforce-addr

做数学运算前把将要使用的内存地址常数送入寄存器.它可能和`**-fforce-mem**'一样产生较好的 目标码.我乐意倾听不同意见.

-fomit-frame-pointer

对于不需要帧指针(frame pointer)的函数,不要在寄存器中保存帧指针.这样能够避免保存,设置和恢复 帧指针的指令;同时对许多函数提供一个额外的寄存器. *但是在大多数机器上将无法调试*.

某些机器上,如 Vax,这个选项无效,因为标准调用序列自动处理帧指针,通过假装不存在而不保存任何东西.机器描述宏 **FRAME_POINTER_REQUIRED** 控制目标机是否支持这个选项.

-finline-functions

把所有简单的函数集成进调用者.编译器探索式地决定哪些函数足够简单,值得这种集成.

如果集成了所有给定函数的调用,而且函数声明为 **static**,那么一般说来 GCC 有权不按汇编代码输出函数.

-fcaller-saves

允许在寄存器里分配数值,但是这个方案通常受到各个函数调用的冲击,因此 GCC 生成额外的代码,在函数调用的 前后保存和复原寄存器内容.仅当生成代码看上去优于反之结果时才实现这样的分配.

某些机器上该选项默认为允许,通常这些机器没有调用保护寄存器代替使用.

-fkeep-inline-functions

即使集成了某个函数的所有调用,而且该函数声明为 **static**,仍然输出这个函数一个独立的,运行时可调用 的版本.

-fno-function-cse

不要把函数地址存入寄存器;让调用固定函数的指令显式给出函数地址.

这个选项产生效率较低的目标码,但是如果不用这个选项,某些不寻常的 hack,改变汇编器的输出,可能因优化而带来 困惑.

-fno-peephole

禁止任何机器相关的 peephole 优化.

-ffast-math

这个选项出于速度优化,允许 GCC 违反某些 ANSI 或 IEEE 规则/规格.例如,它允许编译器假设 **sqrt** 函数的参数是非负数.

这个选项不被任何`**-O**'选项打开,因为对于严格依靠 IEEE 或 ANSI 规则/规格实现的数学函数,程序可能 会产生错误的结果.

下列选项控制特定的优化. `**-O2**'选项打开下面的大多数优化项,除了`**-funroll-loops**'和 `**-funroll-all-loops**'项.

而`**-O**'选项通常打开`**-fthread-jumps**'和`**-fdelayed-branch**' 优化项,但是特定的机

器上的默认优化项有可能改变.

如果特别情况下非常需要``微调''优化,你可以使用下面的选项.

-fstrength-reduce

执行循环强度缩小(loop strength reduction)优化,并且消除重复变量.

-fthread-jumps

执行优化的地点是,如果某个跳转分支的目的地存在另一个条件比较,而且该条件比较包含在前一个比较语句之内,那么 执行优化.根据条件是 true 或者 false,前面那条分支重定向到第二条分支的目的地或者紧跟在第二条分支后面.

-funroll-loops

执行循环展开(loop unrolling)优化.仅对循环次数能够在编译时或运行时确定的循环实行.

-funroll-all-loops

执行循环展开(loop unrolling)优化.对所有循环实行.通常使程序运行的更慢.

-fcse-follow-jumps

在公共子表达式消元(common subexpression elimination)的时候,如果没有其他路径到达某个跳转的 目的地,就扫过这条 jump 指令.例如,如果 CSE 遇到带有 **e lse** 从句的 **if** 语句,当条件测试为 false 时, CSE 就跟在 jump 后面.

-fcse-skip-blocks

它类似于`**-fcse-follow-jumps**'选项,但是 CSE 跟在条件跳转后面,条件跳转跳过了 语句块(block).如果 CSE 遇到一条简单的 **if** 语句,不带 else 从句, `**-fcse-skip-blocks**'选项将导致 CSE 跟在 **if** 产生的跳转后面.

-frerun-cse-after-loop

执行循环优化后,重新进行公共子表达式消元.

-felide-constructors

如果看上去合理就省略构造子(仅 C++).根据这个选项,对于下面的代码, GNU C++直接从调用 **foo** 初始化 **y**,而无需通过临时变量:

```
A foo (); A y = foo ();
```

如果没有这个选项, GNU C++首先通过调用类型 **A** 合适的构造子初始化 **y**;然后把 **f oo** 的结果赋给临时变量;最后,用临时变量替换`**y**'的初始值.

ANSI C++标准草案规定了默认行为(`**-fno-elide-constructors**').如果程序的构造子存在 副效应, `**-felide-constructors**'选项能够使程序有不同的表现,因为可能忽略一些构造子的调用.

-fexpensive-optimizations

执行一些相对开销较大的次要优化.

-fdelayed-branch

如果对目标机支持这个功能,它试图重新排列指令,以便利用延迟分支(delayed branch)指令后面的指令空隙.

-fschedule-insns

如果对目标机支持这个功能,它试图重新排列指令,以便消除因数据未绪造成的执行停顿.这可以帮助浮点运算或内存访问 较慢的机器调取指令,允许其他指令先执行,直到调取指令或浮点运算完成.

-fschedule-insns2

类似于`**-fschedule-insns**'选项,但是在寄存器分配完成后,需要一个额外的指令调度过程.对于 寄存器数目相对较少,而且取内存指令大于一个周期的机器,这个选项特别有用.

目标机选项(TARGET OPTION)

缺省情况下, GNU CC 编译出本机类型的目标码.然而也可以把他安装成交叉编译器, 为其他机型编译程序.事实上,针对不同的目标机,可以同时安装 GNU CC 相应的配置.然后用`**-b**'选项指定 目标机种.

顺便提一下,新版本和旧版本的 GNU CC 可以共存.其中一个版本(可能是最新的那个)为缺省版本,但是有时候你希望使用 其他版本.

-b *machine*

参数 *machine* 指出编译的目标机种.这个选项用于安装为交叉编译器的 GNU CC.

参数 *machine* 的值和配置 GNU CC 交叉编译器时设置的机器类型一样.例如,如果交叉编译器配置有 `**configure** i386v',意思是编译 80386 上的 System V 目标码,

那么你可以通过`**-b i386v**'运行交叉编译器.

如果没有指定`**-b**'选项,通常指编译本机目标码.

-V *version*

参数 *version* 指出运行哪个版本的 GNU CC.这个选项用于安装了多个版本的 GCC.例如,如果 *version* 是`**2.0**',意味着运行 GNU CC 2.0 版.

如果没有指定`**-V**'选项,缺省版本取决于 GNU CC 的安装方式,一般说来推荐使用通用版本.

机器相关选项(MACHINE DEPENDENT OPTION)

每一种目标机型都有自己的特别选项,这些选项用`**-m** '开关引导,选择不同的硬件型号或配置---例如, 68010 还是 68020,有没有浮点协处理器.通过指定选项,安装 编译器的一个版本能够为所有的型号或配置进行编译.

此外,编译器的某些配置支持附加的特殊选项,通常是为了在命令行上兼容这个平台的其他编译器.

下面是针对 68000 系列定义的`**-m**'选项:

-m68000
-mc68000

输出 68000 的目标码.如果编译器按基于 68000 的系统配置,这个选项就是缺省选项.

-m68020
-mc68020

输出 68020 的目标码(而不是 68000).如果编译器按基于 68020 的系统配置,这个选项就是缺省选项.

-m68881

输出包含 68881 浮点指令的目标码.对于大多数基于 68020 的系统这是缺省选项,除非设置编译器时指定了 **-nfp** .

-m68030

输出 68030 的目标码.如果编译器按基于 68030 的系统配置,这个选项就是缺省选项.

-m68040

输出 68040 的目标码.如果编译器按基于 68040 的系统配置,这个选项就是缺省选项.

-m68020-40

输出 68040 的目标码,但是不使用新指令.生成的代码可以在 68020/68881 上,也可以在 68030 或 68040 上较有效地运行.

-mfpa

输出包含 SUN FPA 浮点指令的目标码.

-msoft-float

输出包含浮点库调用的目标码. *警告:*所需的库不是 GNU CC 的组成部分.一般说来 GCC 使用该机型本地 C 编译器的相应部件,但是作交叉编译时却不能直接使用.你必须自己管理提供合适的函数库用于交叉编译.

-mshort

认为 **int** 类型是 16 位宽,相当于 **short int**.

-mnobitfield

不使用位域(bit-field)指令. `**-m68000**'隐含指定了`**-mnobitfield**'.

-mbitfield

使用位域指令. `**-m68020**'隐含指定了`**-mbitfield**'.如果你使用未改装的 gcc,这就是 默认选项.

-mrtd

采用另一种函数调用约定,函数接受固定数目的参数,用 **rtd** 指令返回,该指令返回时弹出栈内的参数.这个 方法能够使调用者节省一条指令,因为他这里不需要弹出参数.

这种调用约定不兼容 UNIX 的正常调用.因此如果你需要调用 UNIX 编译器编译的库函数,你就不能使用这个选项.

此外,所有参数数量可变地函数必须提供函数原型(包括 **printf**);否则编译器会生成错误的调用代码.

另外,如果调用函数时携带了过多的参数,编译器将生成严重错误的代码. (正常情况

下,多余的参数被安全无害的忽略.)

68010 和 68020 处理器支持 **rtd** 指令,但是 68000 不支持.

下面是针对 VAX 定义的`**-m**'选项:

-munix

禁止输出某些跳转指令(**aobleq** 等等), VAX 的 UNIX 汇编器无法跨越长范围(long ranges) 进行处理.

-mgnu

如果使用 GNU 汇编器,则输出那些跳转指令,

-mg

输出 g-format 浮点数,取代 d-format.

下面是 SPARC 支持的`**-m**'选项开关:

-mfpu

-mhard-float

输出包含浮点指令的目标码.这是缺省选项.

-mno-fpu

-msoft-float

输出包含浮点库调用的目标码. *警告:*没有为 SPARC 提供 GNU 浮点库.一般说来使用该机型本地 C 编译器 的相应部件,但是不能直接用于交叉编译.你必须自己安排,提供用于交叉编译的库函数.

-msoft-float 改变了输出文件中的调用约定;因此只有用这个选项编译*整个*程序才有意义.

-mno-epilogue

-mepilogue

使用**-mepilogue** (缺省)选项时,编译器总是把函数的退出代码放在函数的尾部.任何在函数中间 的退出语句(例如 C 中的 return 语句)将产生出跳转指令指向函数尾

部.

使用**-mno-epilogue** 选项时,编译器尽量在每个函数退出点嵌入退出代码.

-mno-v8

-mv8
-msparclite

这三个选项选择不同种类的 SPARC 系统.

默认情况下(除非特别为 Fujitsu SPARClite 配置), GCC 生成 SPARC v7 目标码.

-mv8 生成 SPARC v8 目标码.他和 v7 目标码唯一的区别是,编译器生成整数乘法和整数除法指令, SPARC v8 支持该指令,而 v7 体系不支持.

-msparclite 生成 SPARClite 目标码.增加了 SPARClite 支持的整数乘法,整数除法单步扫描 (integer divide step and scan (ffs))指令. v7 体系不支持这些指令.

-mcypress

-msupersparc

这两个选项选择处理器型号,针对处理器进行代码优化.

-mcypress 选项(默认项)使编译器对 Cypress CY7C602 芯片优化代码, SparcStation/SparcServer 3xx 系列使用这种芯片.该选项也适用于老式的 SparcStation 1, 2, IPX 等机型..

-msupersparc 选项使编译器对 SuperSparc 处理器优化代码, SparcStation 10, 1000 和 2000 系列使用这种芯片.同时该选项启用完整的 SPARC v8 指令集.

下面是针对 Convex 定义的`**-m**'选项:

-mc1

输出 C1 的目标码.当编译器对 C1 配置时,这是默认选项.

-mc2

输出 C2 的目标码.当编译器对 C2 配置时,这是默认选项.

-margcount

在每个参数列表的前面放置一个参数计数字(argument count word).某些不可移植的 Convex 和 Vax 程序需要这个参数计数字. (调试器不需要他,除非函数带有变长参数列表;这个信息存放在符号表中.)

-mnoargcount

忽略参数计数字.如果你使用未改装的 gcc,这是默认选项.

下面是针对 AMD Am29000 定义的`**-m**'选项:

-mdw

生成的目标码认为 DW 置位,就是说,字节和半字操作由硬件直接支持.该选项是默认选项.

-mnodw

生成的目标码认为 DW 没有置位.

-mbw

生成的目标码认为系统支持字节和半字写操作.该选项是默认选项.

-mnbw

生成的目标码认为系统不支持字节和半字写操作.该选项隐含开启了`**-mnodw**'选项.

-msmall

使用小内存模式,小内存模式假设所有函数的地址位于某个 256 KB 段内,或者所有函数的绝对地址小于 256K.这样 就可以用 **call** 指令代替 **const**, **consth**, **calli** 指令序列.

-mlarge

假设不能使用 **call** 指令;这是默认选项.

-m29050

输出 Am29050 的目标码.

-m29000

输出 Am29000 的目标码.这是默认选项.

-mkernel-registers

生成的目标码引用 **gr64-gr95** 寄存器而不是 **gr96-gr127** 寄存器.该选项可以用于编译 内核代码,内核需要一组全局寄存器,这些全局寄存器和用户模式使用的寄存器完

全无关.

注意,使用这个选项时, `**-f**'选项中的寄存器名字必须是 normal, user-mode, names.

-muser-registers

使用普通全局寄存器集 **gr96-gr127**.这是默认选项.

-mstack-check

在每次堆栈调整后插入一条 **__msp_check** 调用.这个选项常用于内核代码.

下面是针对 Motorola 88K 体系定义的`**-m**'选项:

-m88000

生成的目标码可以在 m88100 和 m88110 上正常工作.

-m88100

生成的目标码在 m88100 上工作的最好,但也可以在 m88110 上运行.

-m88110

生成的目标码在 m88110 上工作的最好,可能不能在 m88100 上运行.

-midentify-revision

在汇编器的输出端包含一条 **ident** 指令,记录源文件名,编译器名字和版本,时标,以及使用的编译选项,

-mno-underscores

在汇编器的输出端,符号名字前面不添加下划线.默认情况是在每个名字前面增加下划线前缀.

-mno-check-zero-division

-mcheck-zero-division

早期型号的 88K 系统在除零操作上存在问题,特定情况下许多机器无法自陷.使用这些选项可以避免包含(或可以 显明包含)附加的代码,这些代码能够检查除零错,发送例外信号. GCC 所有 88K 的配置默认使用 `**-mcheck-zero-division**'选项.

-mocs-debug-info

-mno-ocs-debug-info

包含(或忽略)附加的调试信息(关于每个栈架结构中寄存器的使用), 88Open Object Compatibility Standard, ``OCS'',对此信息做了说明. GDB 不需要这些额外信息. DG/UX, SVr4,和 Delta 88 SVr3.2 的默认配置是包含调试信息,其他

88k 机型的默认配置是忽略这个信息.

-mocs-frame-position

-mno-ocs-frame-position

强制(或不要求)把寄存器值存储到栈架结构中的指定位置(按 OCS 的说明). DG/UX, Delta88 SVr3.2 和 BCS 的默认配置使用`**-mocs-frame-position**'选项;其他 88k 机型的默认配置是 `**-mno-ocs-frame-position**'.

-moptimize-arg-area

-mno-optimize-arg-area

控制如何在堆栈结构中存储函数参数. `**-moptimize-arg-area**'节省空间,但是有可能宕掉某些 调试器(不是 GDB). `**-mno-optimize-arg-area**'证实比标准选项好.默认情况下 GCC 不优化参数域.

-mshort-data-

num 通过和 **r0** 关联,产生较小的数据引用(data reference),这样就可以用单指令调入 一个数值(而不是平常的双指令).用户通过选项中的 *num* 控制改变哪种数据引用.例如,如果你指定了 `**-mshort-data-512**',那么受影响的数据引用是小于 512 字节的数据移动. **-mshort-data-***num* 选项对大于 64K 的 *num* 无效.

-mserialize-volatile

-mno-serialize-volatile

产生,或不产生代码来保证对易变内存访问的结果一致.

对于常用的处理器子型号, GNU CC 始终默认保证这种一致性.如何实现结果一致取决于处理器子型号.

m88100 处理器不对内存引用重新安排,因此访问结果始终一致.如果使用了`**-m88100**'选项, GNU CC 不产生任何针对结果一致的特别指令.

m88110 处理器的内存引用顺序并不始终符合指令请求的引用顺序.特别是某条读取指令可能在先前的存储指令之前执行. 多处理器环境下,乱序访问扰乱了易变内存访问的结果一致.因此当使用`**-m88000**'或`**-m88110**' 选项时, GNU CC 在适当的时候产生特别的指令迫使执行顺序正确.

这些用于保证一致性的额外代码有可能影响程序的性能.如果你确认能够安全地放弃这

种保证,你可以使用 `**-mno-serialize-volatile**'选项.

如果你使用`**-m88100**'选项,但是需要在 m88110 处理器上运行时的结果一致,你应该加上 `**-mserialize-volatile**'选项.

-msvr4

-msvr3

打开(`**-msvr4**')或关闭(`**-msvr3**')和 System V 第四版(SVr4)相关的 编译器扩展.效果如下:

*

输出哪种汇编语法(你可以使用`**-mversion-03.00**'选项单独选择).

*

`**-msvr4**'使 C 预处理器识别`**#pragma weak**'指令

*

`**-msvr4**'使 GCC 输出额外的声明指令(declaration directive),用于 SVr4.

除了 SVr4 配置, `**-msvr3**'是所有 m88K 配置的默认选项.

-mtrap-large-shift
-mhandle-large-shift

包含一些指令,用于检测大于 31 位的位移(bit-shift);根据相应的选项,对这样的位移发出自陷 (trap)或执行适当的处理代码.默认情况下, GCC 对大位移不做特别处理.

-muse-div-instruction

很早以前的 88K 型号没有(div)除法指令,因此默认情况下 GCC 避免产生这条指令.而这个选项告诉 GCC 该指令是 安全的.

-mversion-03.00

在 DG/UX 配置中存在两种风格的 SVr4.这个选项修改**-msvr4** ,选择 hybrid-COFF 或 real-ELF 风格.其他配置均忽略该选项.

-mwarn-passed-structs

如果某个函数把结构当做参数或结果传递, GCC 发出警告.随着 C 语言的发展,人们已经改变了传递结构的约定, 它往往导致移植问题.默认情况下, GCC 不会发出警告.

下面的选项用于 IBM RS6000:

-mfp-in-toc

-mno-fp-in-toc

控制是否把浮点常量放到内容表(TOC)中,内容表存放所有的全局变量和函数地址.默认情况下, GCC 把浮点常量放到 这里;如果 TOC 溢出, `**-mno-fp-in-toc**'选项能够减少 TOC 的大小,这样就可以避免溢出.

下面的`**-m**'选项用于 IBM RT PC:

-min-line-mul

对于整数乘法使用嵌入代码.这是默认选项.

-mcall-lib-mul

对于整数乘法使用 **lmul$$** .

-mfull-fp-blocks

生成全尺寸浮点数据块,包括 IBM 建议的最少数量的活动空间(scratch space).这是默认选项.

-mminimum-fp-blocks

不要在浮点数据块中包括额外的活动空间.这样就产生较小但是略慢的可执行程序,因为活动空间必须动态分配.

-mfp-arg-in-fpregs

采用不兼容 IBM 调用约定的调用序列,通过浮点寄存器传送浮点参数.注意,如果指定了这个选项, **varargs.h** 和 **stdargs.h** 将无法支持浮点单元.

-mfp-arg-in-gregs

使用正常的调用约定处理浮点参数.这是默认选项.

-mhc-struct-return

通过内存返回大于一个字的结构,而不是通过寄存器.用于兼容 MetaWare HighC (hc)编译器.使用 `**-fpcc-struct-return**'选项可以兼容 Portable C 编译器(pcc).

-mnohc-struct-return

如果可以,通过寄存器返回某些大于一个字的结构.这是默认选项.如果打算兼容 IBM 提供的编译器,请使用 `**-fpcc-struct-return**'或`**-mhc-struct-return**'选

项.

下面的`**-m**'选项用于 MIPS 家族的计算机:

-mcpu=*cpu-type*

生成指令的时候,假设默认的机器类型是 *cpu-type* .默认情况下的 *cpu-type* 是 **default**, GCC 将选取任何机型上都是最长周期时间的指令,这样才能使代码在所有的 MIPS 处理器上以合理 的速度运行. *cpu-type* 的其他选择是 **r2000**, **r3000**, **r4000**,和 **r6000**.虽然选定某个 *cpu-type* 后, GCC 将针对选定的芯片安排对应的工作,但是如果 不指定?? **-mips2** 或**-mips3** 选项,编译器不会输出任何不符合 MIPS ISA (instruction set architecture)一级的代码.

-mips2

输出 MIPS ISA 二级指令(可能的扩展,如平方根指令). **-mcpu=r4000** 或**-mcpu=r6000** 选项必须和**-mips2** 联用.

-mips3

输出 MIPS ISA 三级指令(64 位指令). **-mcpu=r4000** 选项必须和**-mips2** 联用.(译注:疑为-mips3)

-mint64
-mlong64
-mlonglong128

这些选项目前不起作用.

-mmips-as

产生用于 MIPS 汇编器的代码,同时使用 **mips-tfile** 添加普通的调试信息.对于大多数平台这是 默认选项,除了 OSF/1 参考平台,它使用 OSF/rose 目标格式.如果打开了任一个**-ggdb**, **-gstabs**,或**-gstabs+**选项开关, **mips-tfile** 程序就把 stab 封装在 MIPS ECOFF 里面.

-mgas

产生用于 GNU 汇编器的代码.在 OSF/1 参考平台上这是默认选项,它使用 OSF/rose 目标格式.

-mrnames
-mno-rnames

-mrnames 开关选项告诉输出代码使用 MIPS 软件名称说明寄存器,而不是硬件名称(就是说,用 **a0** 代替**$4**). GNU 汇编器不支持**-mrnames** 选项,而 MIPS 汇编器则运行

MIPS C 预处理器处理源文件. **-mno-rnames** 是默认选项.

-mgpopt

-mno-gpopt

-mgpopt 开关选项要求在正文段中把所有的数据声明写到指令前面,使各种 MIPS 汇编器对短类型全局 或静态数据项(short global or static data items)输出单字内存访问而不是双字内存访问.当打开编译优化 时,这是默认功能.

-mstats

-mno-stats

每次处理完非嵌入函数(non-inline function)后, **-mstats** 开关选项使编译器向标准错误文件 输出一行关于程序的统计资料(保存的寄存器数目,堆栈大小,等等).

-mmemcpy

-mno-memcpy

-mmemcpy 开关选项使所有的块移动操作调用适当的 string 函数(**memcpy** 或 **bcopy**),而不是生成嵌入代码.

-mmips-tfile

-mno-mips-tfile

当 MIPS 汇编器生成 **mips-tfile** 文件(用于帮助调试)后, **-mno-mips-tfile** 开关选项阻止编译器使用 **mips-tfile** 后期处理(postprocess)目标文件.不运行 **mips-tfile** 就没有调试器关注的局部变量.另外, **stage2** 和 **stage3** 目标文件将把临时文件名传递给汇编器,嵌在目标文件中,这意味着不比较目标文件是否相同.

-msoft-float

输出包含浮点库调用. *警告:* 所需库不是 GNU CC 的一部分.一般说来使用该机型本地 C 编译器的相应部件, 但是不能直接用于交叉编译,你必须自己安排,提供交叉编译适用的库函数.

-mhard-float

输出包含浮点指令.如果编译器没有被改动,这就是默认选项.

-mfp64

编译器认为状态字的 **FR** 置位(on),也就是说存在 32 64-bit 浮点寄存器,而不是 32 32-bit 浮点寄存器.同时必须打开**-mcpu=r4000** 和**-mips3** 开关.

-mfp32

认为存在 32 32-bit 浮点寄存器.这是默认选项.

-mabicalls

-mno-abicalls

输出(或不输出) **.abicalls**, **.cpload**,和**.cprestore** 伪指令,某些 System V.4 版本用于位置无关代码.

-mhalf-pic

-mno-half-pic

-mhalf-pic 开关选项要求把外部引用的指针放到数据段,并且载入内存,而不放到正文段.该选项目前 不起作用.

-G num

把小于等于 *num* 字节的全局或静态数据放到小的数据段或 bss 段,而不是普通的数据段或 bss 段. 这样汇编器可以输出基于全局指针(**gp** 或**$28**),的单字内存访问指令而非普通的双字指令.默认情况下, 用 MIPS 汇编器时 *num* 是 8,而 GNU 汇编器则为 0. 另外, **-G***num* 选项也被传递 给汇编器和连接器.所有的模块必须在相同的**-G***num* 值下编译.

-nocpp

汇编用户汇编文件(带有\`**.s**'后缀)时,告诉 MIPS 汇编器不要运行预处理器.

下面的\`**-m**'选项用于 Intel 80386 族计算机: **-m486**

-mno-486

控制是否生成对 486 优化的代码.

-msoft-float

输出包含浮点库调用. *警告:* 所需库不是 GNU CC 的一部分.一般说来使用该机型本地 C 编译器的相应部件, 但是不能直接用于交叉编译,你必须自己安排,提供交叉编译适用的库函数.

在函数把浮点返回值放在 80387 寄存器栈的机器上,即使设置了\`**-msoft-float**'选项,也可能会发出 一些浮点操作码.

-mno-fp-ret-in-387

不用 FPU 寄存器返回函数值.

通常函数调用约定把**float**和**double**的返回值放在FPU寄存器中,即使不存在FPU. 这种作法的理念是操作系统应该仿真出FPU.

而`**-mno-fp-ret-in-387**'选项使浮点值通过普通的CPU寄存器返回.

下面的`**-m**'选项用于HPPA族计算机:

-mpa-risc-1-0

生成PA 1.0处理器的目标码.

-mpa-risc-1-1

生成PA 1.1处理器的目标码.

-mkernel

生成适用于内核的目标码.特别要避免**add**指令,它有一个参数是DP寄存器;用**addil** 代替**add**指令.这样可以避免HP-UX连接器的某个严重bug.

-mshared-libs

生成能够连接HP-UX共享库的目标码.该选项还没有实现全部功能,对PA目标默认为关闭.使用这个选项会导致 编译器生成错误的目标码.

-mno-shared-libs

不生成连接HP-UX共享库的目标码.这是PA目标的默认选项.

-mlong-calls

生成的目标码允许同一个源文件中的函数调用,调用点和被调函数的距离可以超过256K之远.不需要打开这个开关选项, 除非连接器给出``branch out of range errors``这样的错误.

-mdisable-fpregs

防止任何情况下使用浮点寄存器.编译内核需要这个选项,内核切换浮点寄存器的执行环境速度非常缓慢.如果打开了这个 开关选项同时试图浮点操作,编译将失败.

-mdisable-indexing

防止编译器使用索引地址模式(indexing address mode).这样在MACH上编译MIG生成的代码时,可以 避免一些非常晦涩的问题.

-mtrailing-colon

在标记定义(label definition)的末尾添加一个冒号(用于ELF汇编器).

下面的`**-m**'选项用于 Intel 80960 族计算机:

-m*cpu-type*

默认机器类型为 *cpu-type* ,使编译器产生对应的指令,地址模式和内存对齐.默认的 *cpu-type* 是 **kb**;其他选择有 **ka**, **mc**, **ca**, **cf**, **sa**,和 **sb**.

-mnumerics

-msoft-float

-mnumerics 开关选项指出处理器不支持浮点指令. **-msoft-float** 开关选项指出不应该认为 机器支持浮点操作.

-mleaf-procedures

-mno-leaf-procedures

企图(或防止)改变叶过程(leaf procedure),使其可被 *bal* 指令以及 *call* 指令调用.对于直接函数调用,如果 *bal* 指令能够被汇编器或连接器替换,这可以产生更有效的代码,但是其他情况下 产生较低效的代码,例如通过函数指针调用函数,或使用了不支持这种优化的连接器.

-mtail-call

-mno-tail-call

执行(或不执行)更多的尝试(除过编译器那些机器无关部分),优化进入分支的尾递归(tail-recursive)调用.你 可能不需要这个,因为检测什么地方无效没有全部完成.默认开关是**-mno-tail-call**.

-mcomplex-addr

-mno-complex-addr

认为(或不认为)在当前的 i960 设备上,值得使用复合地址模式(complex addressing mode).复合地址模式 可能不值得用到 K 系列,但是一定值得用在 C 系列.目前除了 CB 和 CC 处理器,其他处理器上 **-mcomplex-addr** 是默认选项.

-mcode-align

-mno-code-align

把目标码对齐到 8 字节边界上(或者不必),这样读取会快一些.目前只对 C 系列默认打开.

-mic-compat

-mic2.0-compat

-mic3.0-compat

兼容 iC960 v2.0 或 v3.0.

-masm-compat
-mintel-asm

兼容 iC960 汇编器.

-mstrict-align
-mno-strict-align

不允许(或允许)边界不对齐的访问.

-mold-align

使结构对齐(structure-alignment)兼容 Intel 的 gcc 发行版本 1.3 (基于 gcc 1.37).目前 这个选项有点问题,因为**#pragma align 1** 总是作同样的设定,而且无法关掉.

下面的`**-m**'选项用于 DEC Alpha 设备:

-mno-soft-float
-msoft-float

使用(或不使用)硬件浮点指令进行浮点运算.打开**-msoft-float** 时,将使用 `**libgcc1.c**'中的函数执行浮点运算.除非它们被仿真浮点操作的例程替换,或者类似,它们被编译为调用 仿真例程,这些例程将发出浮点操作.如果你为不带浮点操作的 Alpha 编译程序,你必须确保建立了这个库,以便不调用 仿真例程.

注意,不带浮点操作的 Alpha 也要求拥有浮点寄存器.

-mfp-reg
-mno-fp-regs

生成使用(或不使用)浮点寄存器群的目标代码. **-mno-fp-regs** 包含有**-msoft-float** 开关选项.如果不使用浮点寄存器,浮点操作数就象整数一样通过整数寄存器传送,浮点运算结果放到$0 而不是$f0.这是非标准 调用,因此任何带有浮点参数或返回值的函数,如果被**-mno-fp-regs** 开关编译过的目标码调用,它也必须 用这个选项编译.

这个选项的典型用法是建立内核,内核不使用任何浮点寄存器,因此没必要保存和恢复这些寄存器.

下面附加的选项出现在 System V 第四版中,用于兼容这些系统中的其他编译器:

-G

在 SVr4 系统中, **gcc** 出于兼容接受了`**-G**'选项(然后传递给连接器).可是我们建议使用 `**-symbolic**'或`**-shared**'选项,而不在 **gcc** 命令行上出现连接选项.

-Qy

验证编译器用的工具的版本,输出到**.ident** 汇编指令.

-Qn

制止输出端的**.ident** 指令(默认选项).

-YP,*dirs*

对于`**-l**'指定的库文件,只搜索 *dirs*.你可以在 *dirs* 中用冒号隔开各个 目录项.

-Ym,*dir*

在 *dir* 目录中寻找 M4 预处理器.汇编器使用这个选项.

代码生成选项(CODE GENERATION OPTION)

下面的选项和平台无关,用于控制目标码生成的接口约定.

大部分选项以`-f'开始.这些选项拥有确定和否定两种格式; `**-ffoo**'的否定格式是 `**-fno-foo**'.后面的描述将只列举其中的一个格式---非默认的格式.你可以通过添加或去掉 `**no-**'推测出另一个格式.

-fnonnull-objects

假设通过引用(reference)取得的对象不为 null (仅 C++).

一般说来, GNU C++对通过引用取得的对象作保守假设.例如,编译器一定会检查下似代码中的 **a** 不为 null:

```
obj &a = g (); a.f (2);
```

检查类似的引用需要额外的代码,然而对于很多程序是不必要的.如果你的程序不要求这种检查,你可以用 `**-fnonnull-objects**'选项忽略它.

-fpcc-struct-return

函数返回 **struct** 和 **union** 值时,采用和本地编译器相同的参数约定.对于较小的结构, 这种约定的效率偏低,而且很多机器上不能重入;它的优点是允许 GCC 编译的目标码和 PCC 编译的目标码互相调用.

-freg-struct-return

一有可能就通过寄存器返回 **struct** 和 **union** 函数值.对于较小的结构,它比 **-fpcc-struct-return** 更有效率.

如果既没有指定**-fpcc-struct-return** ,也没有指定**-freg-struct-return**, GNU CC 默认使用目标机的标准约定.如果没有标准约定, GNU CC 默认采用**-fpcc-struct-return**.

-fshort-enums

给 **enum** 类型只分配它声明的值域范围的字节数.就是说, **enum** 类型等于大小足够的最小整数类型.

-fshort-double

使 **double** 类型的大小和 **float** 一样.

-fshared-data

要求编译结果的数据和非 **const** 变量是共享数据,而不是私有数据.这种差别仅在某些操作系统上面有意义, 那里的共享数据在同一个程序的若干进程间共享,而私有数据在每个进程内都有副件.

-fno-common

即使未初始化的全局变量也分配在目标文件的 bss 段,而不是把它们当做普通块(common block)建立.这样的 结果是,如果在两个不同的编译结果中声明了同一个变量(没使用 **extern**),连接它们时会产生错误. 这个选项可能有用的唯一情况是,你希望确认程序能在其他系统上运行,而其他系统总是这么做.

-fno-ident

忽略`**#ident**'指令.

-fno-gnu-linker

不要把全局初始化部件(如 C++的构造子和解构子)输出为 GNU 连接器使用的格式(在 GNU 连接器是标准方法的系统 上).当你打算使用非 GNU 连接器的时候可以用这个选项,非 GNU 连接器也需要 **collect2** 程序确保系统连接器 放入构造子(constructor)和解构子(destructor). (GNU CC 的发布包中包含有 **collect2** 程序.)对于*必须*使用 **collect2** 的系统,编译器驱动程序 **gcc** 自动配置为这么做.

-finhibit-size-directive

不要输出**.size** 汇编指令,或其他类似指令,当某个函数一分为二,两部分在内存中距离很远时会引起问题. 当编译`**crtstuff.c**'时需要这个选项;其他情况下都不应该使用.

-fverbose-asm

输出汇编代码时放些额外的注释信息.这个选项仅用于确实需要阅读汇编输出的时候(可能调试编译器自己的时候).

-fvolatile

使编译器认为所有通过指针访问的内存是易变内存(volatile).

-fvolatile-global

使编译器认为所有的外部和全局变量是易变内存.

-fpic

如果支持这种目标机,编译器就生成位置无关目标码.适用于共享库(shared library).

-fPIC

如果支持这种目标机,编译器就输出位置无关目标码.适用于动态连接(dynamic linking),即使分支需要大范围 转移.

-ffixed-*reg*

把名为 *reg* 的寄存器按固定寄存器看待(fixed register);生成的目标码不应该引用它(除了或许 用作栈指针,帧指针,或其他固定的角色).

reg 必须是寄存器的名字.寄存器名字取决于机器,用机器描述宏文件的 **REGISTER_NAMES** 宏 定义.

这个选项没有否定格式,因为它列出三路选择.

-fcall-used-*reg*

把名为 *reg* 的寄存器按可分配寄存器看待,不能在函数调用间使用.可以临时使用或当做变量使用,生存期 不超过一个函数.这样编译的函数无需保存和恢复 *reg* 寄存器.

如果在可执行模块中,把这个选项说明的寄存器用作固定角色将会产生灾难性结果,如栈指针或帧指针.

这个选项没有否定格式,因为它列出三路选择.

-fcall-saved-*reg*

把名为 *reg* 的寄存器按函数保护的可分配寄存器看待.可以临时使用或当做变量使用,它甚至能在函数间 生存.这样编译的函数会保存和恢复使用中的 *reg* 寄存器.

如果在可执行模块中,把这个选项说明的寄存器用作固定角色将会产生灾难性结果,如栈指针或帧指针.

另一种灾难是用这个选项说明的寄存器返回函数值.

这个选项没有否定格式,因为它列出三路选择.

PRAGMAS

GNU C++支持两条`**#pragma**'指令使同一个头文件有两个用途:对象类的接口定义, 对象类完整的内容定义.

#pragma interface

(仅对 C++)在定义对象类的头文件中,使用这个指令可以节省大部分采用该类的目标文件的大小.一般说来,某些信息 (内嵌成员函数的备份副件,调试信息,实现虚函数的内部表格等)的本地副件必须保存在包含类定义的各个目标文件中.使用这个 pragma 指令能够避免这样的复制.当编译中引用包含`**#pragma interface**'指令的头文件时,就 不会产生这些辅助信息(除非输入的主文件使用了`**#pragma implementation**'指令).作为替代,目标文件 将包含可被连接时解析的引用(reference).

#pragma implementation

#pragma implementation "*objects***.h"**

(仅对 C++)如果要求从头文件产生完整的输出(并且全局可见),你应该在主输入文件中使用这条 pragma.头文件 中应该依次使用`**#pragma interface**'指令.在 implementation 文件中将产生全部内嵌成员函数 的备份,调试信息,实现虚函数的内部表格等.

如果`**#pragma implementation**'不带参数,它指的是和源文件有相同基本名的包

含文件;例如, `**allclass.cc**'中, `**#pragma implementation**'等于`**#pragma implementation allclass.h**'.如果某个 implementation 文件需要从多个头文件引入代码,就应该 使用这个字符串参数.

不可能把一个头文件里面的内容分割到多个 implementation 文件中.

文件(FILE)

file.c C 源文件
file.h C 头文件(预处理文件)
file.i 预处理后的 C 源文件
file.C C++源文件
file.cc C++源文件
file.cxx C++源文件
file.m Objective-C 源文件
file.s 汇编语言文件
file.o 目标文件
a.out 连接的输出文件
TMPDIR/cc* 临时文件
LIBDIR/cpp 预处理器
LIBDIR/cc1 C 编译器
LIBDIR/cc1plus C++编译器
LIBDIR/collect 某些机器需要的连接器前端(front end)程序
LIBDIR/libgcc.a GCC 子例程(subroutine)库
/lib/crt[01n].o 启动例程(start-up)
LIBDIR/ccrt0 C++的附加启动例程
/lib/libc.a 标准 C 库,另见 intro (3)
/usr/include **#include** 文件的标准目录
LIBDIR/include **#include** 文件的标准 gcc 目录
LIBDIR/g++-include **#include** 文件的附加 g++目录

LIBDIR 通常为**/usr/local/lib/***machine*/*version*.

TMPDIR 来自环境变量 **TMPDIR** (如果存在,缺省为**/usr/tmp** ,否则为 **/tmp**).

另见(SEE ALSO)

cpp(1), as(1), ld(1), gdb(1), adb(1), dbx(1), sdb(1).
info 中 `**gcc**', `**cpp**', `**as**', `**ld**',和`**gdb**'的条目.
Using and Porting GNU CC (for version 2.0), Richard M. Stallman; *The C Preprocessor*, Richard M. Stallman; *Debugging with GDB: the GNU Source-Level Debugger*, Richard M. Stallman 和 Roland H. Pesch; *Using as: the GNU Assembler*, Dean Elsner, Jay Fenlason & friends; *ld: the GNU linker*, Steve Chamberlain 和 Roland Pesch.

BUGS

关于报告差错的指导请查阅 GCC 手册.

版权(COPYING)

作者(AUTHORS)

关于 GNU CC 的奉献者请查阅 GUN CC 手册.

[中文版维护人]

徐明<xuming@users.sourceforge.net>

[中文版最新更新]

2003/05/13 第一版

《中国 Linux 论坛 man 手册页翻译计划》

http://cmpp.linuxforum.net/

Index

This document was created by man2html, using the manual pages.
Time: GMT, January 14, 2004